

Lawrence Livermore National Laboratory

HYPRE: High Performance Preconditioners

BOUT++ Workshop, September 15, 2011



Robert D. Falgout

Center for Applied Scientific Computing

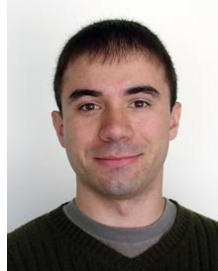
The *hypr* Team



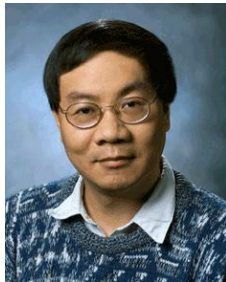
Allison Baker



Rob Falgout



Tzanio Kolev



Charles Tong



Panayot Vassilevski



Ulrike Yang

Former

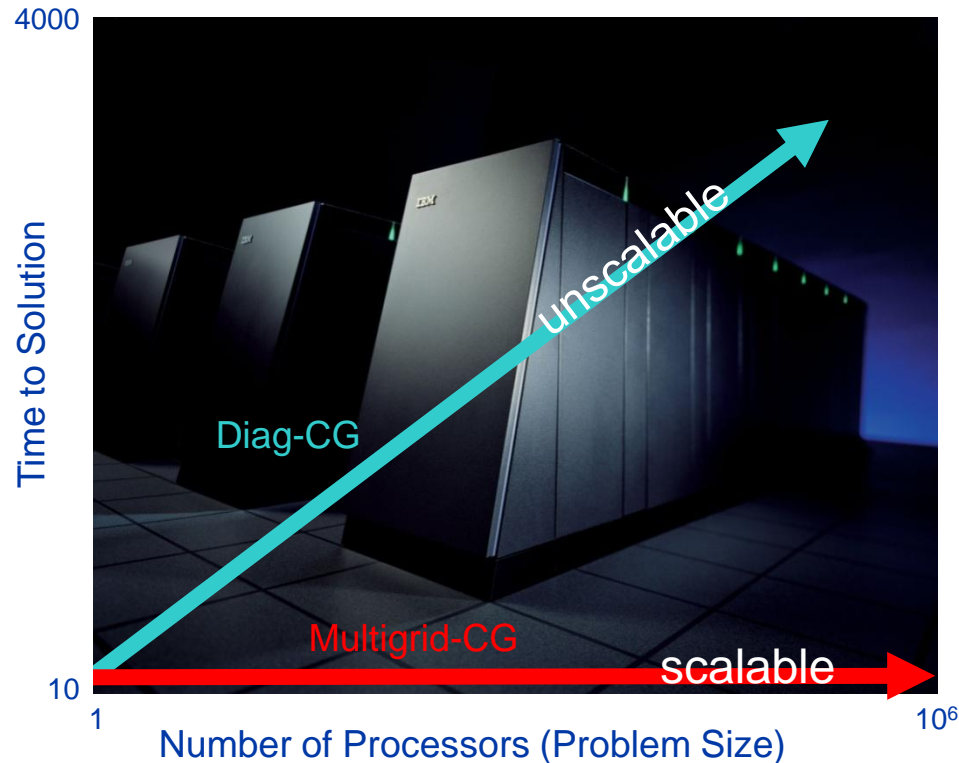
- Chuck Baldwin
- Guillermo Castilla
- Edmond Chow
- Andy Cleary
- Noah Elliott
- Van Henson
- Ellen Hill
- David Hysom
- Jim Jones
- Mike Lambert
- Barry Lee
- Jeff Painter
- Tom Treadway
- Deborah Walker

<http://www.llnl.gov/CASC/hypr/>

Outline

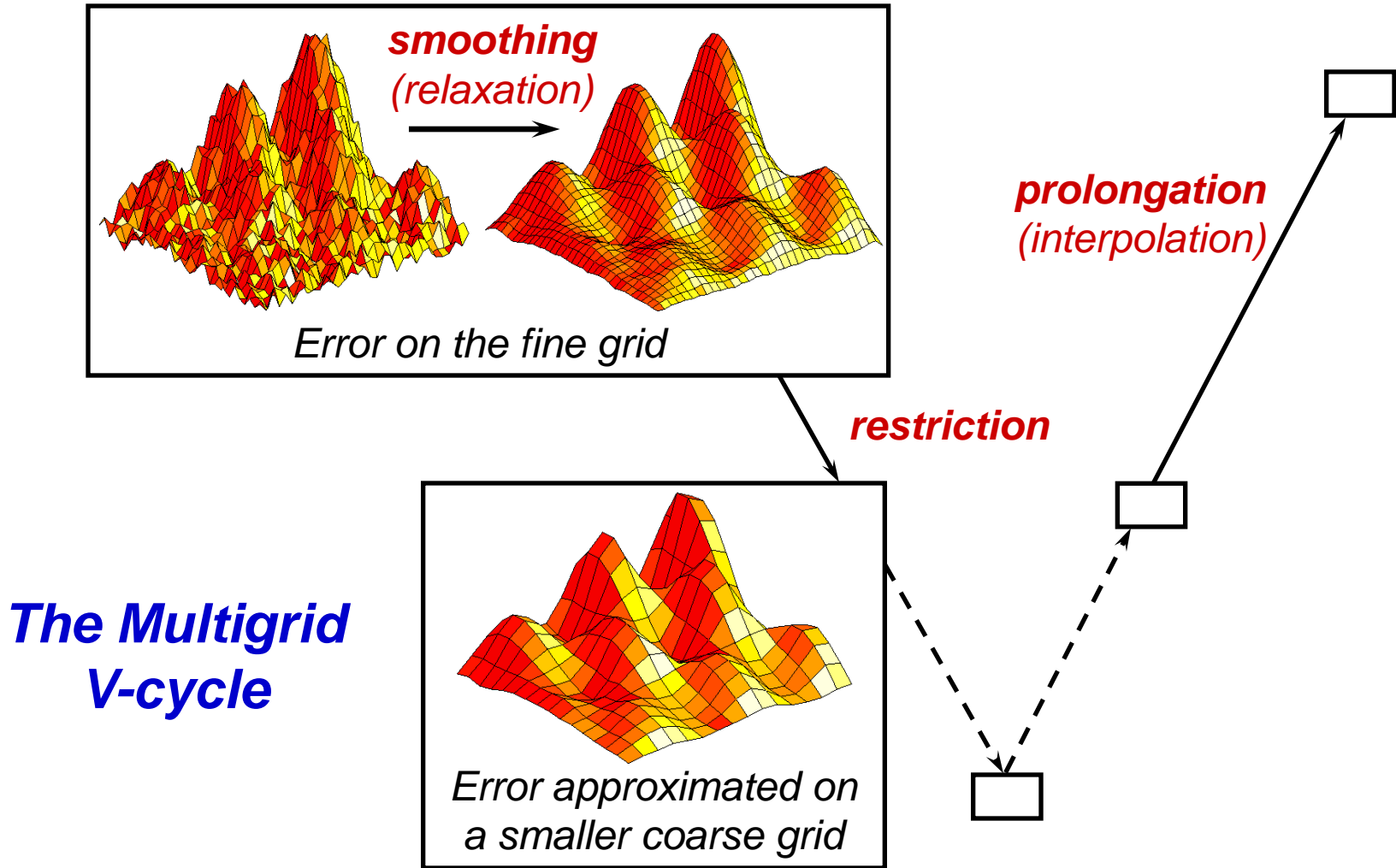
- Introduction / Motivation
- Getting Started / Linear System Interfaces
 - Structured-Grid Interface (`Struct`)
 - Semi-Structured-Grid Interface (`SStruct`)
 - Finite Element Interface (`FEI`)
 - Linear-Algebraic Interface (`IJ`)
- Solvers and Preconditioners
- Additional Information

Multigrid linear solvers are optimal ($O(N)$ operations), and hence have good scaling potential



- Weak scaling – want constant solution time as problem size grows in proportion to the number of processors

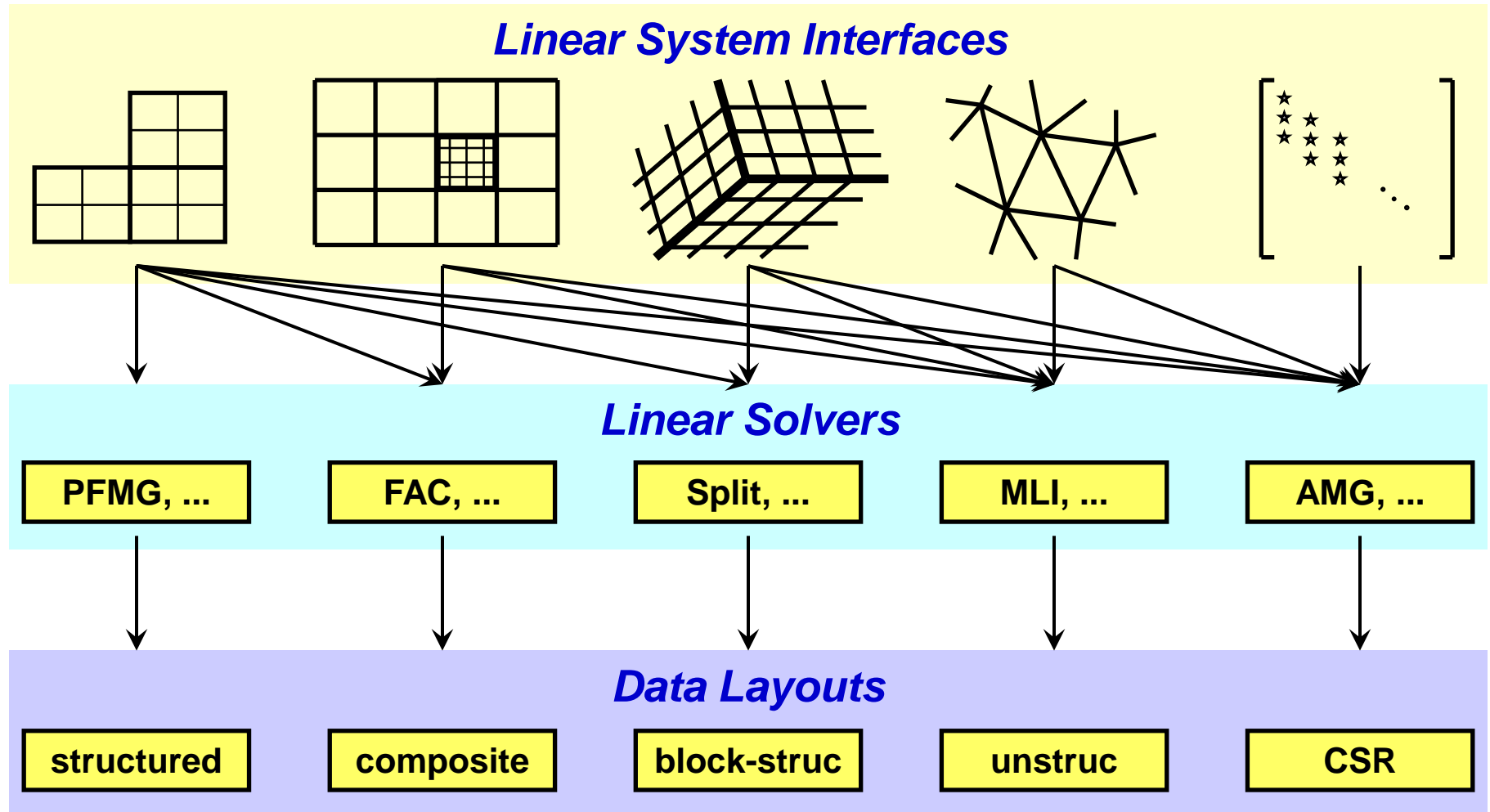
Multigrid uses a sequence of coarse grids to accelerate the fine grid solution



Getting Started

- **Before writing your code:**
 - choose a linear system interface
 - choose a solver / preconditioner
 - choose a matrix type that is compatible with your solver / preconditioner and system interface
- **Now write your code:**
 - build auxiliary structures (e.g., grids, stencils)
 - build matrix/vector through system interface
 - build solver/preconditioner
 - solve the system
 - get desired information from the solver

(Conceptual) linear system interfaces are necessary to provide “best” solvers and data layouts

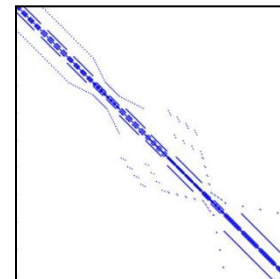
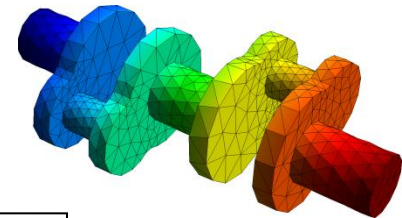
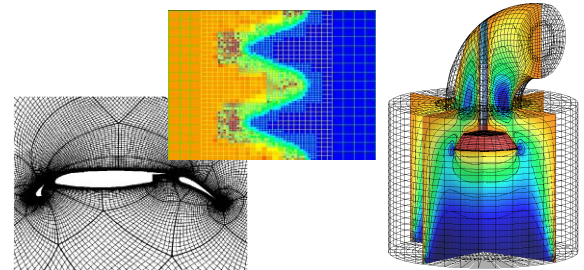
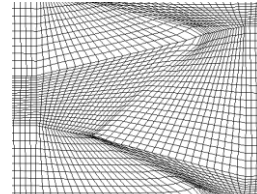


Why multiple interfaces? The key points

- Provides natural “views” of the linear system
- Eases some of the coding burden for users by eliminating the need to map to rows/columns
- Provides for more efficient (scalable) linear solvers
- Provides for more effective data storage schemes and more efficient computational kernels

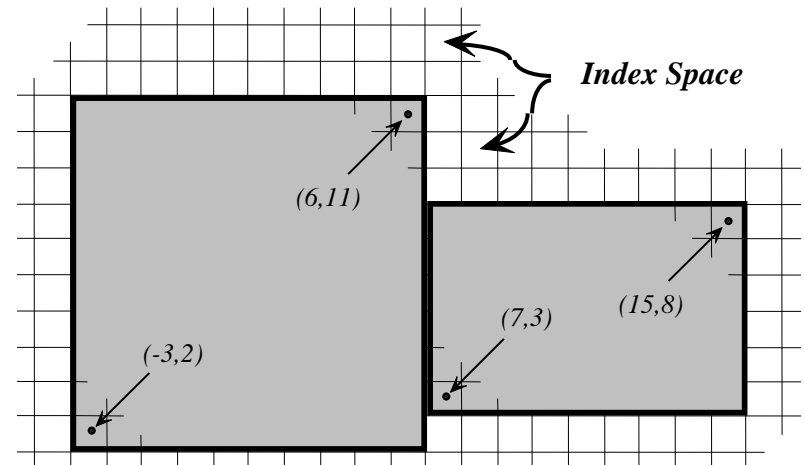
Currently, *hypr* supports four system interfaces

- Structured-Grid ([Struct](#))
 - *logically rectangular grids*
- Semi-Structured-Grid ([SStruct](#))
 - *grids that are mostly structured*
- Finite Element ([FEI](#))
 - *unstructured grids with finite elements*
- Linear-Algebraic ([IJ](#))
 - *general sparse linear systems*
- More about each next...



Structured-Grid System Interface (Struct)

- Appropriate for scalar applications on structured grids with a fixed stencil pattern
- Grids are described via a global d -dimensional *index space* (singles in 1D, tuples in 2D, and triples in 3D)
- A *box* is a collection of cell-centered indices, described by its “lower” and “upper” corners
- The scalar grid data is always associated with cell centers (unlike the more general SStruct interface)

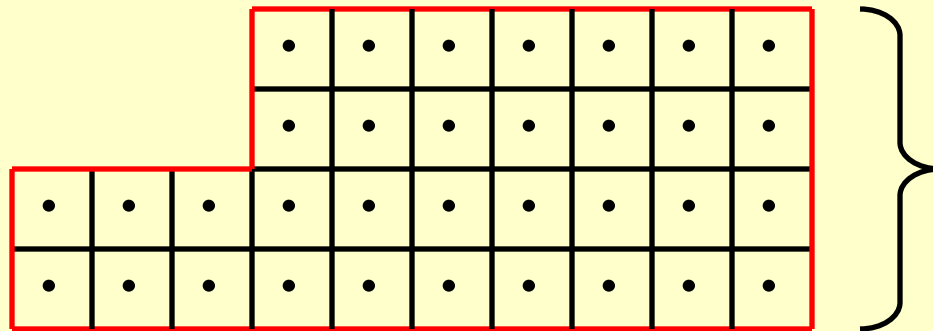


Structured-Grid System Interface (Struct)

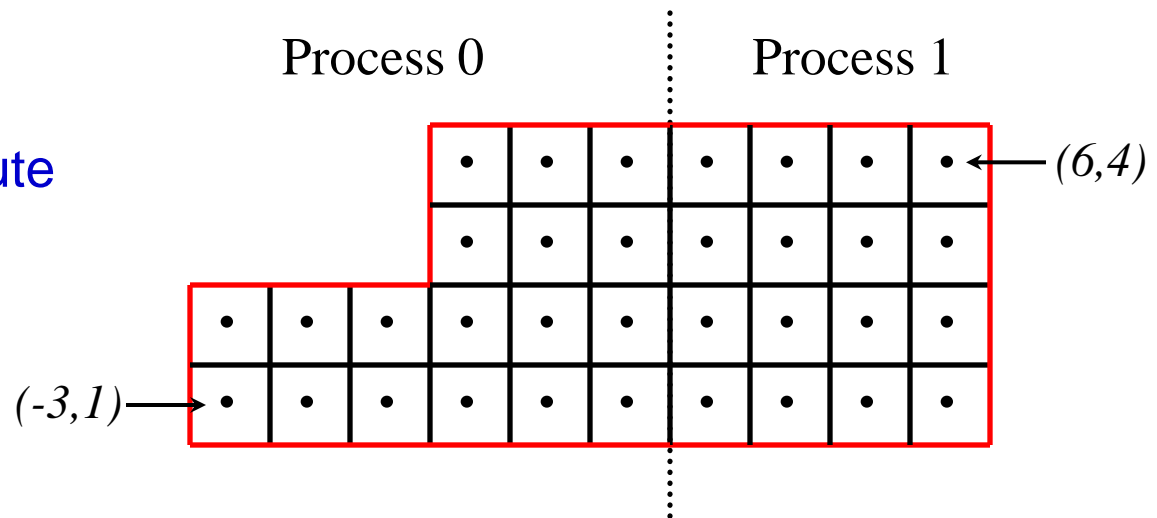
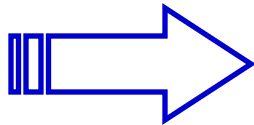
- There are four basic steps involved:
 - set up the `Grid`
 - set up the `Stencil`
 - set up the `Matrix`
 - set up the right-hand-side `Vector`
- Consider the following 2D Laplacian problem

$$\begin{cases} -\nabla^2 u = f & \text{in the domain} \\ u = g & \text{on the boundary} \end{cases}$$

Structured-grid finite volume example:

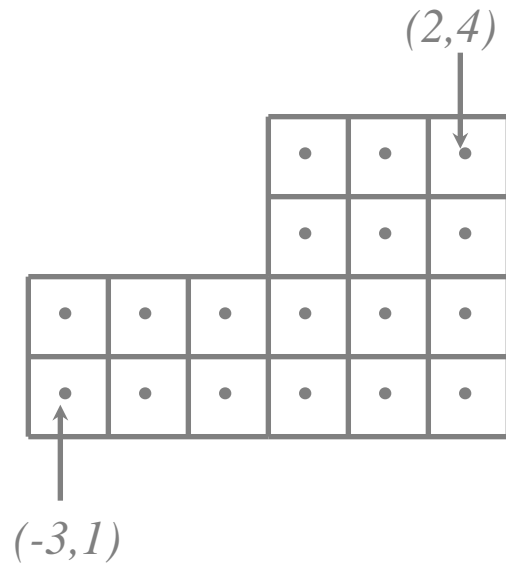


Partition and distribute



Structured-grid finite volume example:

Setting up the grid on process 0

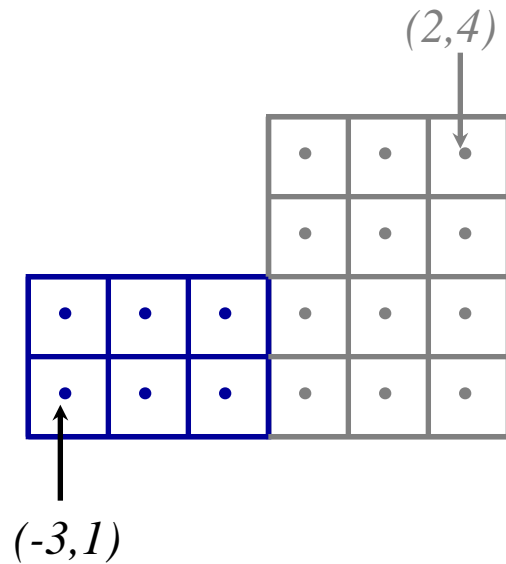


Create the grid object

```
HYPRE_StructGrid grid;  
int ndim      = 2;  
  
HYPRE_StructGridCreate(MPI_COMM_WORLD, ndim, &grid);
```

Structured-grid finite volume example:

Setting up the grid on process 0



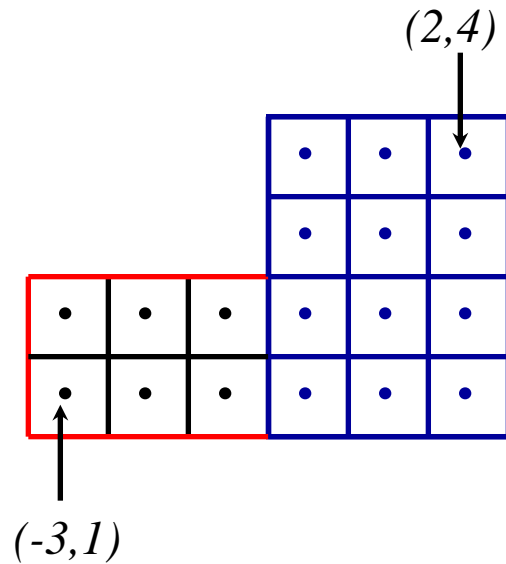
Set grid extents for first box

```
int ilo0[2] = {-3,1};  
int iup0[2] = {-1,2};
```

```
HYPRE_StructGridSetExtents(grid, ilo0, iup0);
```

Structured-grid finite volume example:

Setting up the grid on process 0



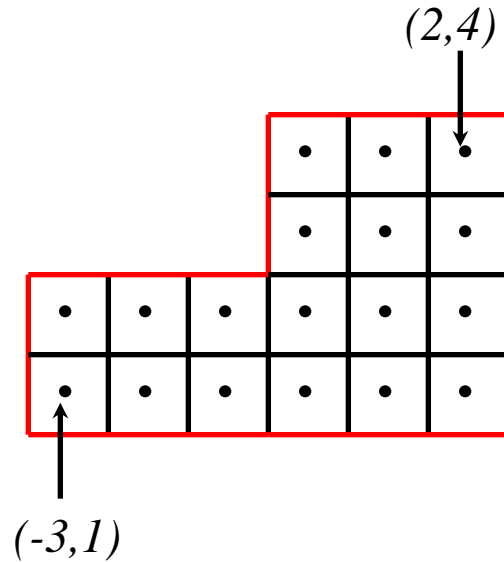
**Set grid extents for
second box**

```
int ilo1[2] = {0,1};  
int iup1[2] = {2,4};
```

```
HYPRE_StructGridSetExtents(grid, ilo1, iup1);
```

Structured-grid finite volume example:

Setting up the grid on process 0

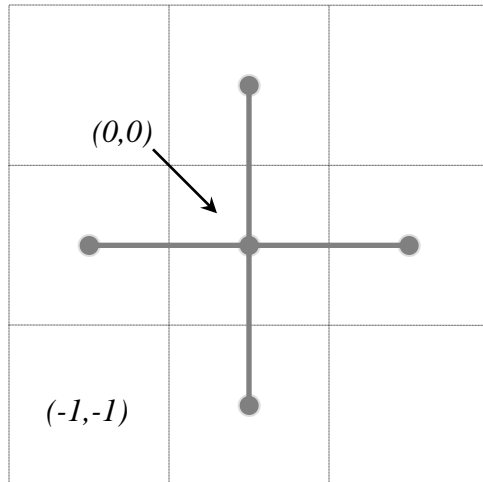


Assemble the grid

```
HYPRE_StructGridAssemble(grid);
```

Structured-grid finite volume example:

Setting up the stencil (all processes)

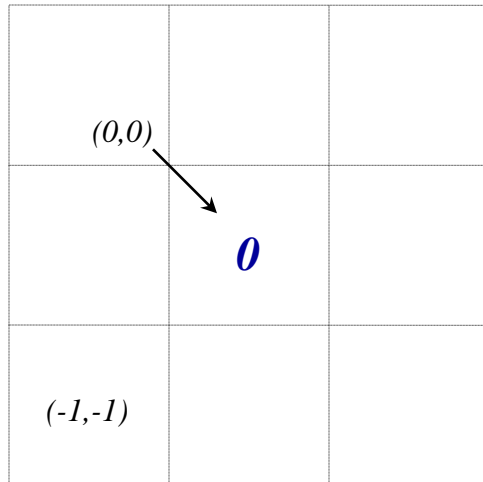


stencil entries		geometries
0	↔	(0, 0)
1	↔	(-1, 0)
2	↔	(1, 0)
3	↔	(0,-1)
4	↔	(0, 1)

Create the stencil object

```
HYPRE_StructStencil stencil;  
int ndim = 2;  
int size = 5;  
  
HYPRE_StructStencilCreate(ndim, size, &stencil);
```

Structured-grid finite volume example: Setting up the stencil (all processes)



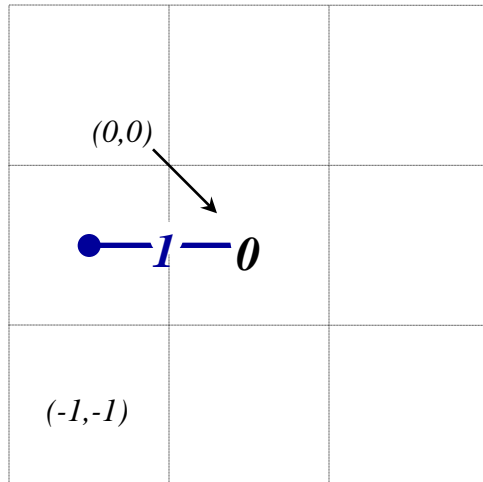
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0,-1)	
	4	↔	(0, 1)	

Set stencil entries

```
int entry = 0;  
int offset[2] = {0,0};
```

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

Structured-grid finite volume example: Setting up the stencil (all processes)



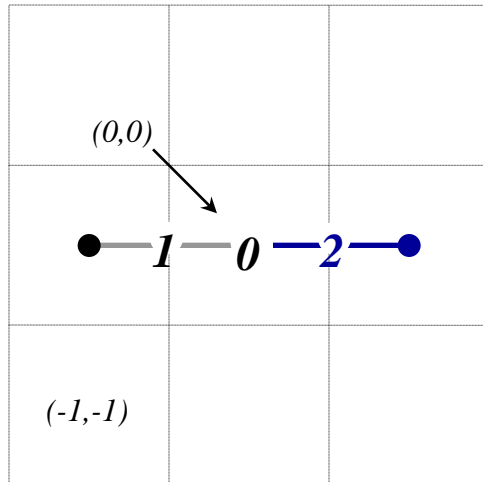
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0, -1)	
	4	↔	(0, 1)	

Set stencil entries

```
int entry = 1;  
int offset[2] = {-1, 0};
```

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

Structured-grid finite volume example: Setting up the stencil (all processes)



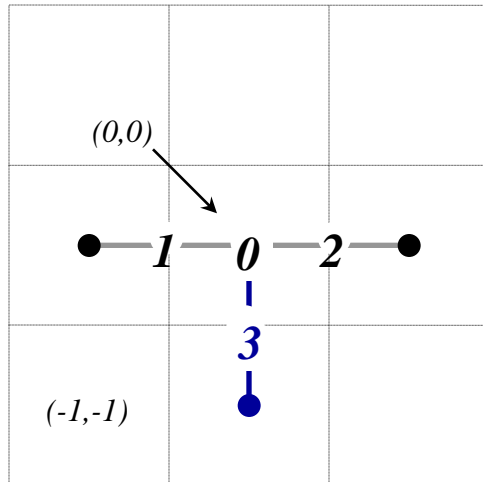
stencil entries		geometries
0	↔	(0, 0)
1	↔	(-1, 0)
2	↔	(1, 0)
3	↔	(0,-1)
4	↔	(0, 1)

Set stencil entries

```
int entry = 2;  
int offset[2] = {1,0};
```

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

Structured-grid finite volume example: Setting up the stencil (all processes)



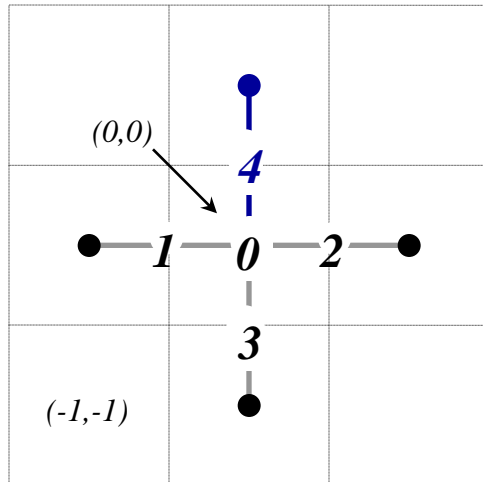
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0,-1)	
	4	↔	(0, 1)	

Set stencil entries

```
int entry = 3;  
int offset[2] = {0,-1};
```

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

Structured-grid finite volume example: Setting up the stencil (all processes)



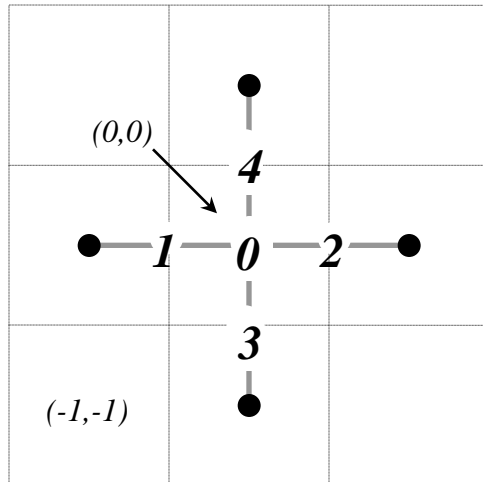
stencil entries		geometries
0	↔	(0, 0)
1	↔	(-1, 0)
2	↔	(1, 0)
3	↔	(0, -1)
4	↔	(0, 1)

Set stencil entries

```
int entry = 4;  
int offset[2] = {0,1};
```

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

Structured-grid finite volume example: Setting up the stencil (all processes)

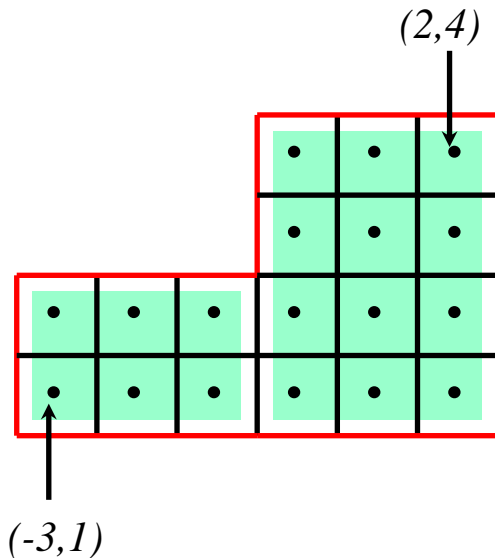


stencil entries	0	↔	$(0, 0)$	geometries
	1	↔	$(-1, 0)$	
	2	↔	$(1, 0)$	
	3	↔	$(0, -1)$	
	4	↔	$(0, 1)$	

That's it!
**There is no assemble
routine**

Structured-grid finite volume example :

Setting up the matrix on process 0



$$\begin{pmatrix} & \text{S4} & \\ \text{S1} & \text{S0} & \text{S2} \\ & \text{S3} & \end{pmatrix} = \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}$$

```
HYPRE_StructMatrix A;
double  vals[24] = {4, -1, 4, -1, ...};
int  nentries = 2;
int  entries[2] = {0,3};

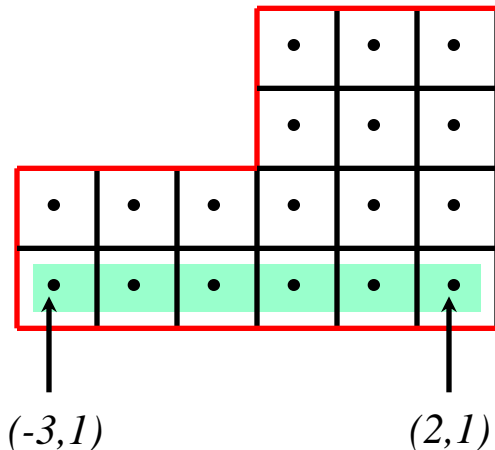
HYPRE_StructMatrixCreate (MPI_COMM_WORLD,
    grid, stencil, &A);
HYPRE_StructMatrixInitialize (A);

HYPRE_StructMatrixSetBoxValues (A,
    ilo0, iup0, nentries, entries, vals);
HYPRE_StructMatrixSetBoxValues (A,
    ilo1, iup1, nentries, entries, vals);

/* set boundary conditions */
...
HYPRE_StructMatrixAssemble (A);
```

Structured-grid finite volume example :

Setting up the matrix bc's on process 0



$$\begin{pmatrix} & \text{S4} & \\ \text{S1} & \text{S0} & \text{S2} \\ & \text{S3} & \end{pmatrix} = \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & 0 & \end{pmatrix}$$

```
int  ilo[2] = {-3, 1};
int  iup[2] = { 2, 1};
double  vals[6] = {0, 0, ...};
int nentries = 1;

/* set interior coefficients */
...

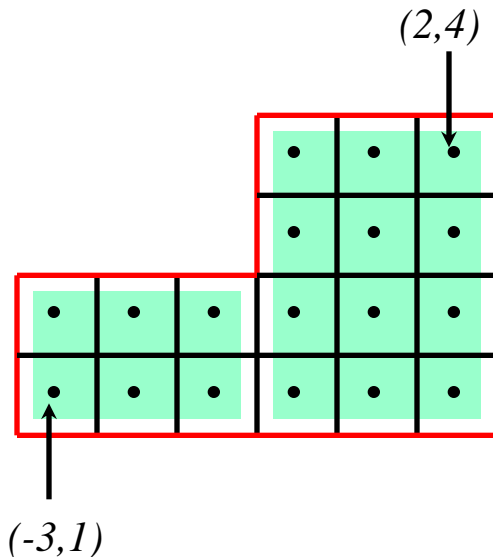
/* implement boundary conditions */
...

i = 3;
HYPRE_StructMatrixSetBoxValues(A,
    ilo, iup, nentries, &i, vals);

/* complete implementation of bc's */
...
```

A structured-grid finite volume example :

Setting up the right-hand-side vector on process 0



```
HYPRE_StructVector b;  
double vals[12] = {0, 0, ...};  
  
HYPRE_StructVectorCreate(MPI_COMM_WORLD,  
    grid, &b);  
HYPRE_StructVectorInitialize(b);  
  
HYPRE_StructVectorSetBoxValues(b,  
    ilo0, iup0, vals);  
HYPRE_StructVectorSetBoxValues(b,  
    ilo1, iup1, vals);  
  
HYPRE_StructVectorAssemble(b);
```

Symmetric Matrices

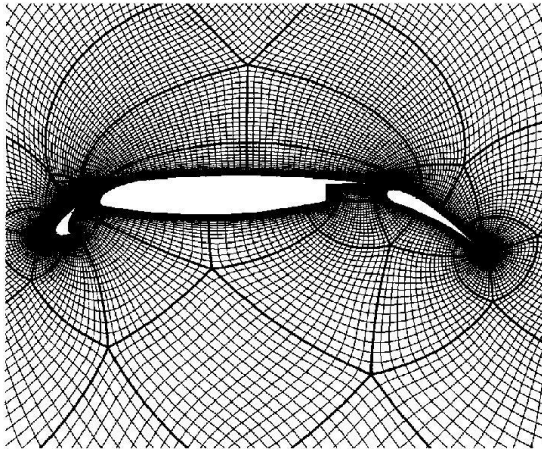
- Some solvers support symmetric storage
- Between `Create()` and `Initialize()`, call:
`HYPRE_StructMatrixSetSymmetric(A, 1);`
- For best efficiency, only set half of the coefficients

$$\begin{bmatrix} (0,1) \\ (0,0) \ (1,0) \end{bmatrix} \Leftrightarrow \begin{bmatrix} s2 \\ s0 \ s1 \end{bmatrix}$$

- This is enough info to recover the full 5-pt stencil

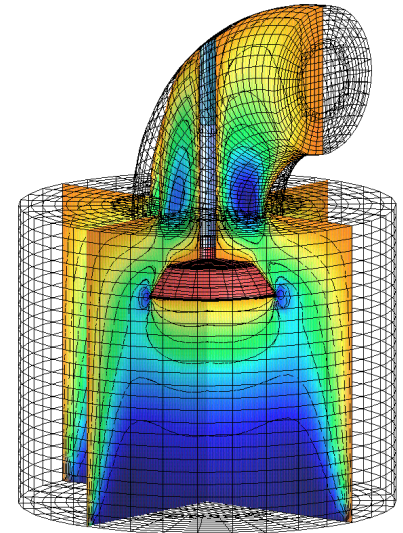
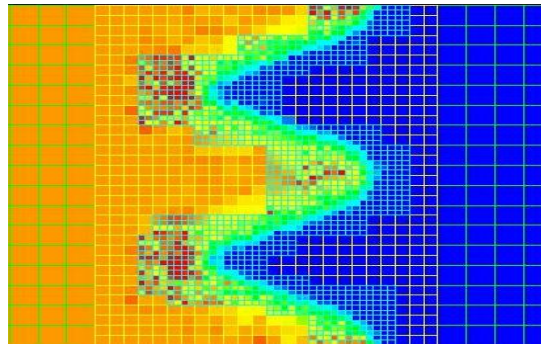
Semi-Structured-Grid System Interface (SStruct)

- Allows more general grids:
 - Grids that are mostly (but not entirely) structured
 - Examples: *block-structured grids*, *structured adaptive mesh refinement grids*, *overset grids*



Block-Structured

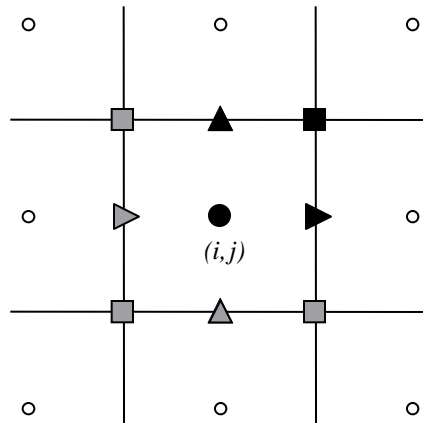
*Adaptive Mesh
Refinement*



Overset

Semi-Structured-Grid System Interface (SStruct)

- Allows more general PDE's
 - Multiple variables (system PDE's)
 - Multiple variable types (cell centered, face centered, vertex centered, ...)



Variables are referenced by the abstract cell-centered index to the left and down

Semi-Structured-Grid System Interface (SStruct)

- The SStruct grid is composed out of structured grid *parts*
- The interface uses a *graph* to allow nearly arbitrary relationships between part data
- The graph is constructed from stencils or **finite element stiffness matrices (new)** plus additional data-coupling information set either
 - directly with `GraphAddEntries()`, or
 - by relating parts with `GridSetNeighborPart()` and `GridSetSharedPart()` **(new)**
- We will consider three examples:
 - block-structured grid using stencils
 - star-shaped grid with finite elements **(new)**
 - structured adaptive mesh refinement

Semi-Structured-Grid System Interface (SStruct)

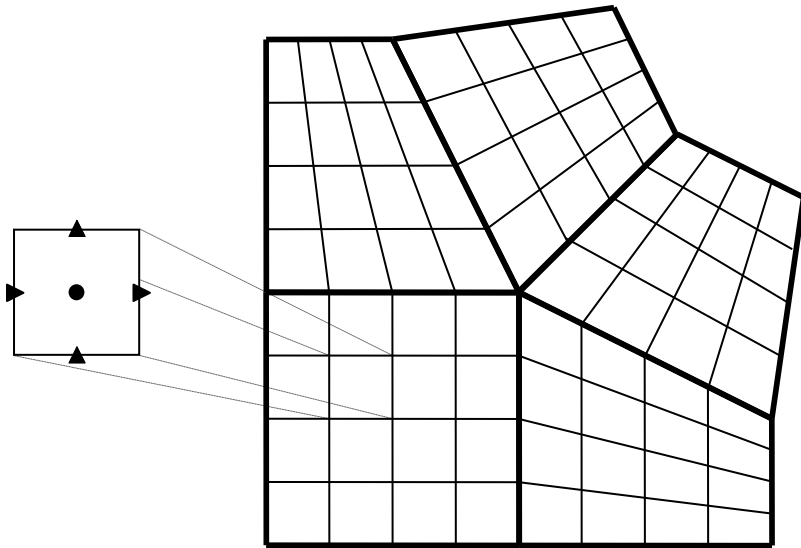
- There are five basic steps involved:
 - set up the `Grid`
 - set up the `Stencils`
 - set up the `Graph`
 - set up the `Matrix`
 - set up the right-hand-side `Vector`

Block-structured grid example (SStruct)

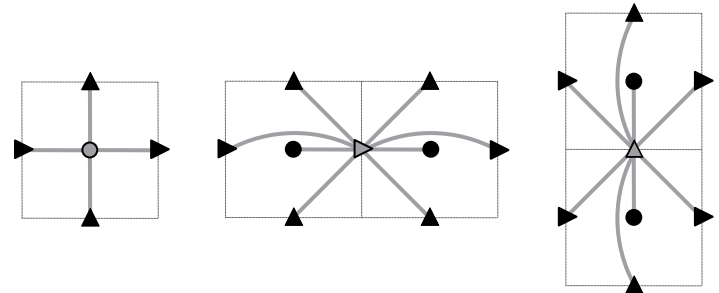
- Consider the following block-structured grid discretization of the diffusion equation

$$-\nabla \cdot \mathbf{K} \nabla u + \sigma u = f$$

A block-structured grid with
3 variable types

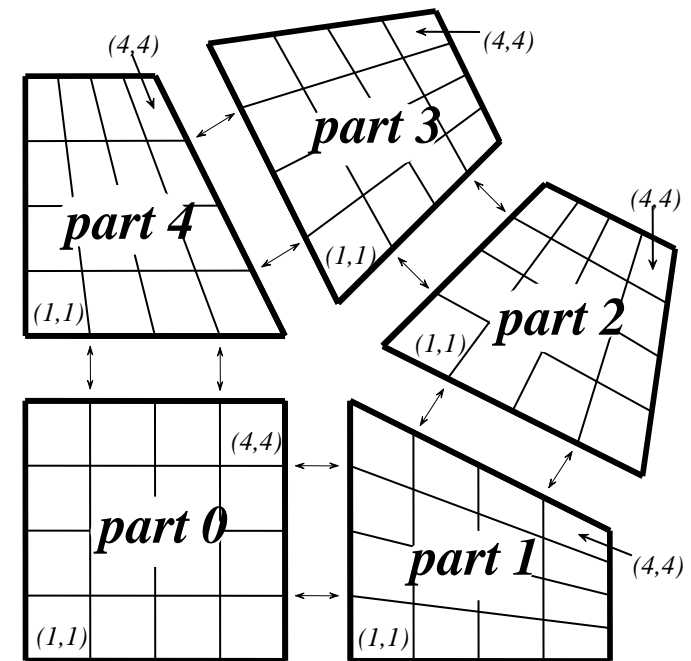


The 3 discretization stencils



Block-structured grid example (SStruct)

- The `Grid` is described via 5 logically-rectangular parts
- We assume 5 processes such that process p owns part p (user defines the distribution)
- Relationship between parts is set with `GridSetNeighborPart()`
- We consider the interface calls made by process 3



New finite element (FEM) style interface for SStruct as an alternative to stencils

- Beginning with *hypr* version 2.6.0b
- `GridSetSharedPart()` is similar to `SetNeighborPart`, but allows one to specify shared cells, faces, edges, or vertices
- `GridSetFEMOrdering()` sets the ordering of the unknowns in an element (always a cell)
- `GraphSetFEM()` indicates that an FEM approach will be used to set values instead of a stencil approach
- `GraphSetFEMSparsity()` sets the nonzero pattern for the stiffness matrix
- `MatrixAddFEMValues()` and `VectorAddFEMValues()`
- See examples: `ex13.c`, `ex14.c`, and `ex15.c`

Finite Element (FEM) example (SStruct)

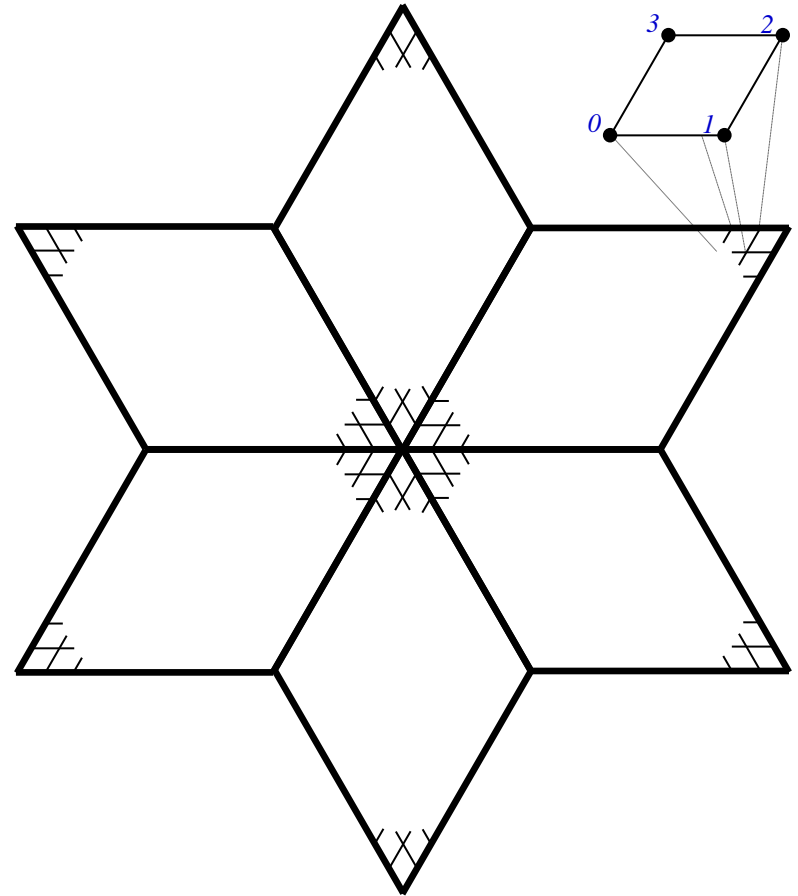
- FEM nodal discretization of the Laplace equation on a star-shaped domain

$$\begin{cases} -\nabla^2 u = 1 & \text{in } \Omega \\ u = 0 & \text{on } \Gamma \end{cases}$$

- FEM stiffness matrix

$$\begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} 4-k & -1 & -2+k & -1 \\ -1 & 4+k & -1 & -2-k \\ -2+k & -1 & 4-k & -1 \\ -1 & -2-k & -1 & 4+k \end{pmatrix} \end{matrix} \alpha$$

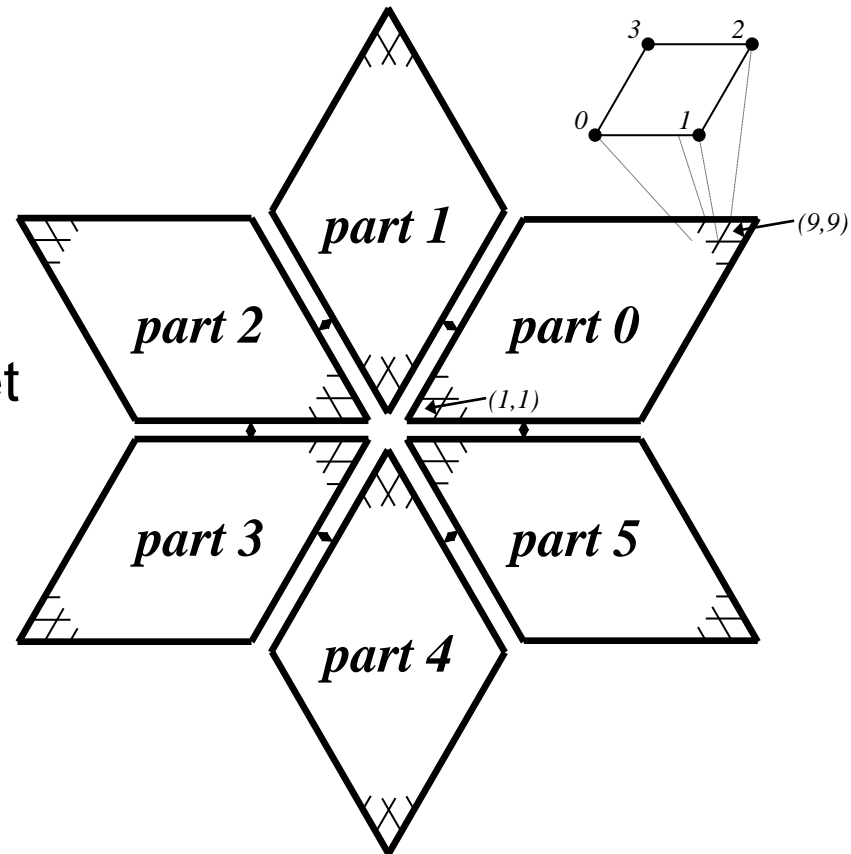
$$\alpha = (6 \sin(\gamma))^{-1}, \quad k = 3 \cos(\gamma), \quad \gamma = \pi/3$$



See example code
ex14.c

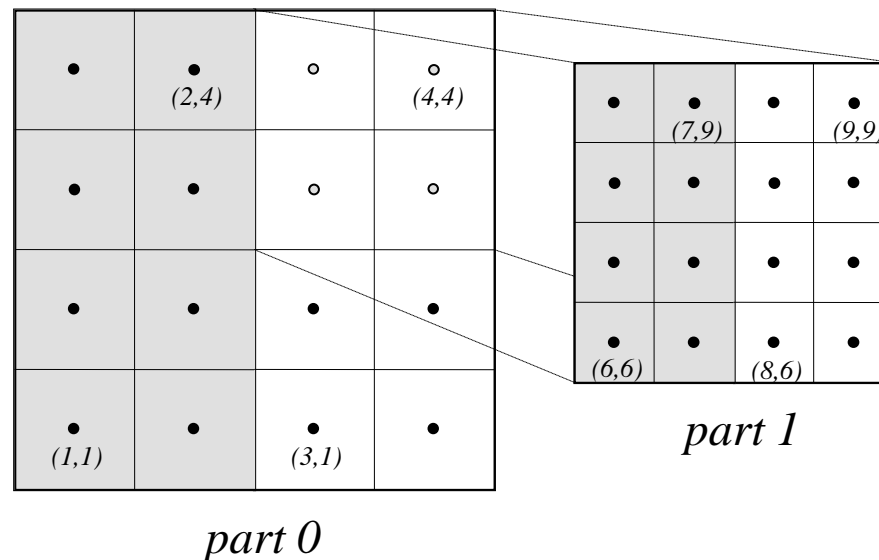
FEM example (SStruct)

- The `Grid` is described via 6 logically-rectangular parts
- We assume 6 processes, where process p owns part p
- Relationship between parts is set with `GridSetSharedPart()`
- The `Matrix` is assembled from stiffness matrices (no stencils)
- We consider the interface calls made by process 0



Structured AMR example (SStruct)

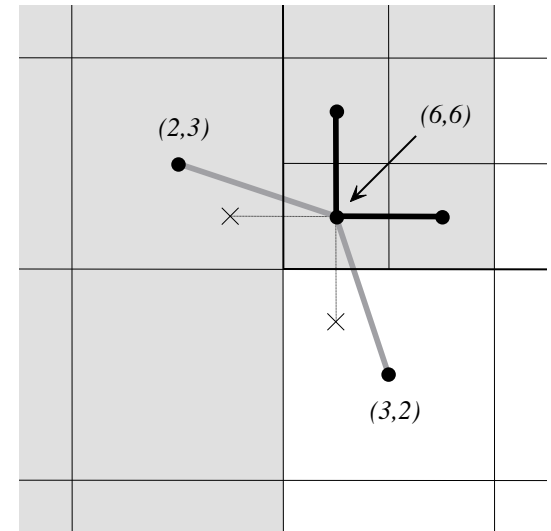
- Consider a simple cell-centered discretization of the Laplacian on the following structured AMR grid



- Each AMR grid level is defined as a separate part
- Assume 2 processes with shaded regions on process 0 and unshaded regions on process 1

Structured AMR example (SStruct)

- The **grid** is constructed using straightforward calls to the routines `HYPRE_SStructGridSetExtents()` and `HYPRE_SStructGridSetVariables()` as in the previous block-structured grid example
- The **graph** is constructed from a cell-centered stencil plus additional *non-stencil entries* at coarse-fine interfaces
- These non-stencil entries are set one variable at a time using `HYPRE_SStructGraphAddEntries()`



Building different matrix/vector storage formats with the SStruct interface

- Efficient preconditioners often require specific matrix/vector storage schemes
- Between `Create()` and `Initialize()`, call:
`HYPRE_SStructMatrixSetObjectType(A, HYPRE_PARCSR);`
- After `Assemble()`, call:
`HYPRE_SStructMatrixGetObject(A, &parcsr_A);`
- Now, use the `ParCSR` matrix with compatible solvers such as BoomerAMG (algebraic multigrid)

Finite Element Interface (FEI)

- The FEI interface is designed for finite element discretizations on unstructured grids
- The interface supports C++ only
- See the following for detailed information on using it

R. L. Clay et al. An annotated reference guide to the Finite Element Interface (FEI) Specification, Version 1.0. Sandia National Laboratories, Livermore, CA, Technical Report SAND99-8229, 1999.

- There is a brief description in *hypr* user's manual and an example code

Linear-Algebraic System Interface (IJ)

- The IJ interface provides access to general sparse-matrix solvers, but not specialized solvers
- This is a “traditional” linear solver interface similar to what is in PETSc, Trilinos, etc.
- There are two basic steps involved:
 - set up the `Matrix`
 - set up the right-hand-side `Vector`

Current solver / preconditioner availability via *hypr*'s linear system interfaces

Data Layouts		System Interfaces			
	Solvers	Struct	SStruct	FEI	IJ
Structured	Jacobi	✓	✓		
	SMG	✓	✓		
	PFMG	✓	✓		
Semi-structured	Split		✓		
	SysPFMG		✓		
	FAC		✓		
	Maxwell		✓		
Sparse matrix	AMS		✓	✓	✓
	BoomerAMG		✓	✓	✓
	MLI		✓	✓	✓
	ParaSails		✓	✓	✓
	Euclid		✓	✓	✓
	PILUT		✓	✓	✓
	PCG	✓	✓	✓	✓
Matrix free	GMRES	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	Hybrid	✓	✓	✓	✓

Setup and use of solvers is largely the same (see *Reference Manual for details*)

- Create the solver

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- Set parameters

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- Prepare to solve the system

```
HYPRE_SolverSetup(solver, A, b, x);
```

- Solve the system

```
HYPRE_SolverSolve(solver, A, b, x);
```

- Get solution info out via system interface

```
HYPRE_StructVectorGetValues(struct_x, index,  
values);
```

- Destroy the solver

```
HYPRE_SolverDestroy(solver);
```

Solver example: SMG-PCG

```
/* define preconditioner (one symmetric V(1,1)-cycle) */
HYPRE_StructSMGCreate(MPI_COMM_WORLD, &precond);
HYPRE_StructSMGSetMaxIter(precond, 1);
HYPRE_StructSMGSetTol(precond, 0.0);
HYPRE_StructSMGSetZeroGuess(precond);
HYPRE_StructSMGSetNumPreRelax(precond, 1);
HYPRE_StructSMGSetNumPostRelax(precond, 1);

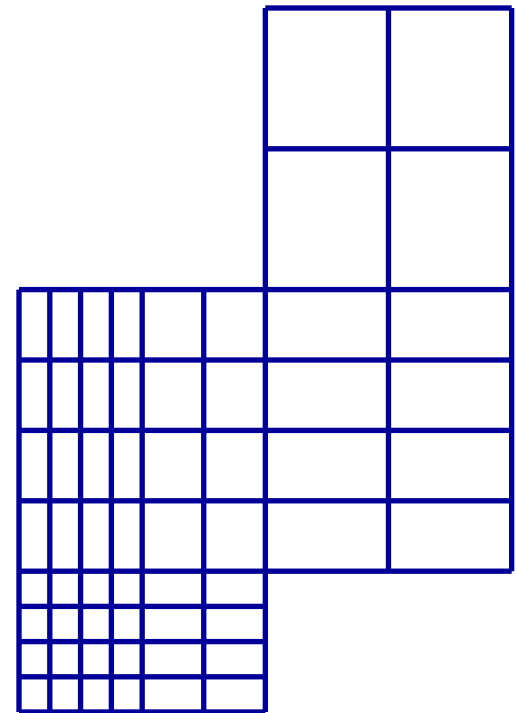
HYPRE_StructPCGCreate(MPI_COMM_WORLD, &solver);
HYPRE_StructPCGSetTol(solver, 1.0e-06);

/* set preconditioner */
HYPRE_StructPCGSetPrecond(solver,
    HYPRE_StructSMGSolve, HYPRE_StructSMGSetup, precond);

HYPRE_StructPCGSetup(solver, A, b, x);
HYPRE_StructPCGSolve(solver, A, b, x);
```

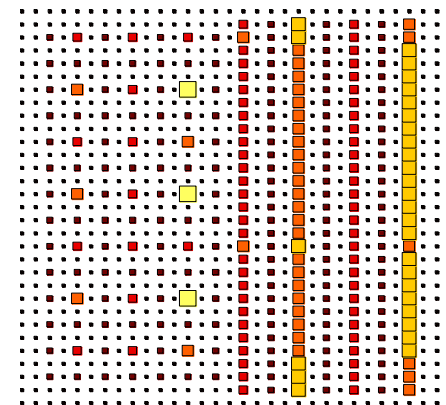
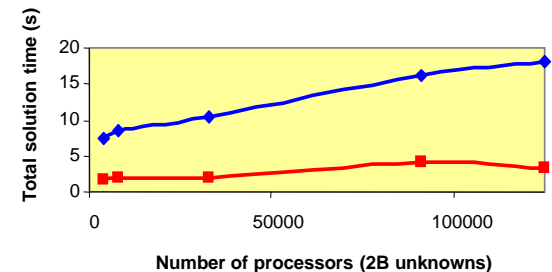
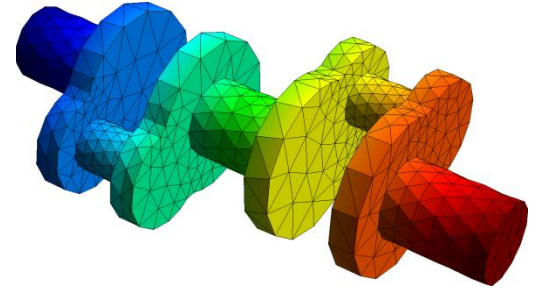
SMG and PFMG are semicoarsening multigrid methods for structured grids

- Interface: `Struct`, `SStruct`
- Matrix Class: `Struct`
- SMG uses plane smoothing in 3D, where each plane “solve” is effected by one 2D V-cycle
- SMG is very robust
- PFMG uses simple pointwise smoothing, and is less robust
- **Constant-coefficient versions!**



BoomerAMG is an algebraic multigrid method for unstructured grids

- Interface: `SStruct`, `FEI`, `IJ`
- Matrix Class: `ParCSR`
- Originally developed as a general matrix method (i.e., assumes given only A , x , and b)
- Various coarsening, interpolation and relaxation schemes
- Automatically coarsens “grids”
- Can solve systems of PDEs if additional information is provided



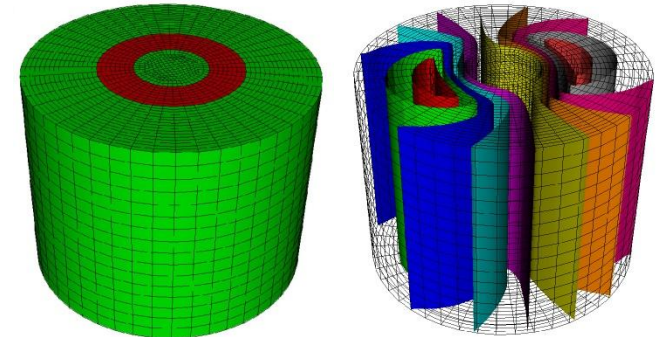
AMS is an auxiliary space Maxwell solver for unstructured grids

- Interface: `SStruct`, `FEI`, `IJ`
- Matrix Class: `ParCSR`

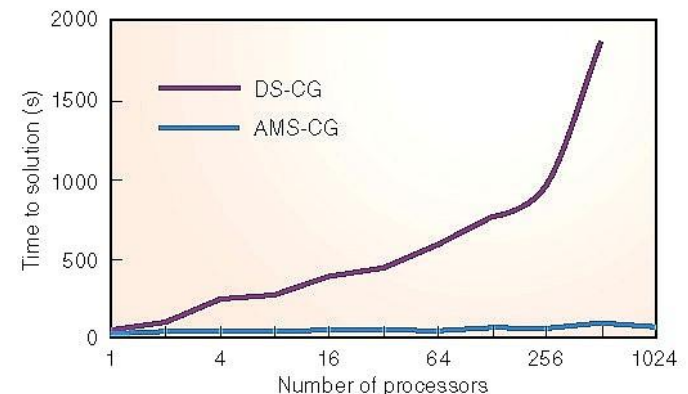
- Solves definite problems:

$$\nabla \times \alpha \nabla \times E + \beta E = f, \quad \alpha > 0, \beta \geq 0$$

- Requires additional gradient matrix and mesh coordinates
- Variational form of Hiptmair-Xu
- Employs BoomerAMG
- Only for FE discretizations



**Copper wire in air,
conductivity jump of 10^6**

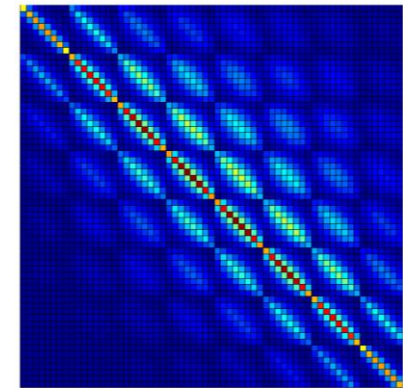


25x faster on 80M unknowns

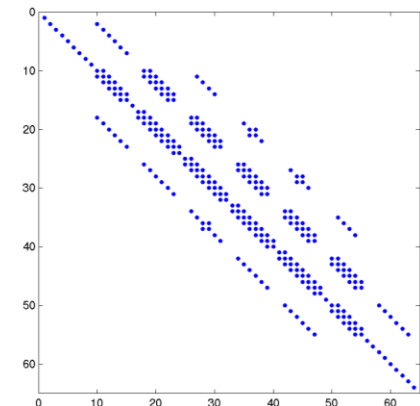
ParaSAILS is an approximate inverse method for sparse linear systems

- Interface: `SStruct`, `FEI`, `IJ`
- Matrix Class: `ParCSR`
- Approximates the inverse of A by a sparse matrix M by minimizing the Frobenius norm of $I - AM$
- Uses graph theory to predict good sparsity patterns for M

Exact inverse



Approx inverse



Euclid is a family of Incomplete LU methods for sparse linear systems

- Interface: `SStruct`, `FEI`, `IJ`
- Matrix Class: `ParCSR`
- Obtains scalable parallelism via local and global reorderings
- Good for unstructured problems



<http://www.cs.odu.edu/~pothen/Software/Euclid>

Getting the code

- To get the code, go to

<http://www.llnl.gov/CASC/hypre/>

- User's / Reference Manuals can be downloaded directly
- A short form must be filled out (just for our own records)

Building the library

- Usually, *hypre* can be built by typing `configure` followed by `make`
- Configure supports several options (for usage information, type '`configure --help`'):
 - '`configure --enable-debug`' - turn on debugging
 - '`configure --with-openmp`' - use openmp
 - '`configure --with-CFLAGS=...`' - set compiler flags
- **Release now includes example programs!**

Calling *hypre* from Fortran

- C code:

```
HYPRE_IJMatrix A;  
int            nvalues, row, *cols;  
double        *values;  
  
HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values);
```

- Corresponding Fortran code:

```
integer*8      A  
integer        nvalues, row, cols(MAX_NVALUES)  
double precision values(MAX_NVALUES)  
  
call HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values)
```

Reporting bugs, requesting features, general usage questions

- Send email to:

hypre-support@llnl.gov

- We use a tool called Roundup to automatically tag and track issues

Additional comments: using *hypr* with...

■ PETSc

- A limited set of *hypr* solvers (including BoomerAMG) and solver parameters are currently available through PETSc
- It is possible to call *hypr* directly from within PETSc (I think), but this would incur additional overhead
- We are collaborating with the PETSc team, so the two packages should become more interoperable in the future

■ SUNDIALS

- Most of the packages appear to provide a general way of hooking in arbitrary linear solver libraries
- For example, CVODE requires setting up four modules: `linit()`, `lsetup()`, `lsolve()`, and `lfree()`

Thank You!

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.